**Affine Transformations and Transformation Hierarchies in OpenGL**

*Wolfgang Heldrich*

---

**Recap: Properties of Affine Transformations**

**Theorem:**
- The following statements are synonymous
  - A transformation \( T(x) \) is affine, i.e.:
    \[
    x' = T(x) := M \cdot x + t,
    \]
    for some matrix \( M \) and vector \( t \)
  - \( T(x) \) preserves affine combinations, i.e.
    \[
    T(\sum_{i=1}^{n} \alpha_i \cdot x_i) = \sum_{i=1}^{n} \alpha_i \cdot T(x_i), \text{ for } \sum_{i=1}^{n} \alpha_i = 1
    \]
  - \( T(x) \) maps parallel lines to parallel lines

---

**Recap: Properties of Affine Transformations**

**Example:**
- Affine combination of 2 points
  \[
  x = \alpha_1 \cdot x_1 + \alpha_2 \cdot x_2,
  \]
  with \( \alpha_1 + \alpha_2 = 1 \)
  \[
  = (1 - \alpha_2) \cdot x_1 + \alpha_2 \cdot x_2
  \]
  \[
  = x_1 + \alpha_2 \cdot (x_2 - x_1)
  \]

---

**Recap: Properties of Affine Transformations**

**Definition:**
- A convex combination is an affine combination where all the weights \( \alpha_i \) are positive
- Note: this implies \( 0 \leq \alpha_i \leq 1 \), \( i=1 \ldots n \)

---

**Recap: Properties of Affine Transformations**

**Example:**
- Convex combination of 3 points
  \[
  x = \alpha_1 \cdot x_1 + \beta_1 \cdot x_2 + \gamma \cdot x_3,
  \]
  with \( \alpha + \beta + \gamma = 1 \), \( 0 \leq \alpha, \beta, \gamma \leq 1 \)
  - \( \alpha, \beta, \) and \( \gamma \) are called *barycentric coordinates*
Recap: Properties of Affine Transformations

*Preservation of affine combinations:*
- Can compute transformation of every point on line or triangle by simply transforming the control points

\[ \begin{align*}
S(x_1, x_2) &= \begin{pmatrix} x_1' \\ y_1' \\ z_1' \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & 0 \\ m_{21} & m_{22} & m_{23} & 0 \\ m_{31} & m_{32} & m_{33} & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} \\
S(x_1', x_2') &= \begin{pmatrix} x_2' \\ y_2' \\ z_2' \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & 0 \\ m_{21} & m_{22} & m_{23} & 0 \\ m_{31} & m_{32} & m_{33} & 0 \end{pmatrix} \begin{pmatrix} x_1' \\ y_1' \\ z_1' \\ 1 \end{pmatrix} 
\end{align*} \]

Recap: Homogeneous Coordinates

*Homogeneous representation of points:*
- Add an additional component \( w - 1 \) to all points
- All multiples of this vector are considered to represent the same 3D point
- Use square brackets (rather than round ones) to denote homogeneous coordinates (different from text book!)

\[
\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}
\]

Recap: Geometrically in 2D

*Cartesian Coordinates:*

\[ \begin{align*}
&x' = m_{11}x + m_{12}y + m_{13}z + t_x \\
y' = m_{21}x + m_{22}y + m_{23}z + t_y \\
z' = m_{31}x + m_{32}y + m_{33}z + t_z \\
&[1, 0, 0, 1] 
\end{align*} \]

Recap: Geometrically in 2D

*Homogeneous Coordinates:*

\[ \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}
\]

Recap: Homogeneous Matrices

*Affine Transformations:

\[
\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & 0 \\ m_{21} & m_{22} & m_{23} & 0 \\ m_{31} & m_{32} & m_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}
\]

Recap: Homogeneous Matrices

*Combining the two matrices into one:

\[
\begin{pmatrix} x'' \\ y'' \\ z'' \\ w'' \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & t_x \\ m_{21} & m_{22} & m_{23} & t_y \\ m_{31} & m_{32} & m_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix}
\]

\[
\begin{pmatrix} x'' \\ y'' \\ z'' \\ w'' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}
\]
Recap: Homogeneous Transformations

Notes:
- A composite transformation is now just the product of a few matrices.
- Rather than multiply each point sequentially with 3 matrices, first multiply the matrices, then multiply each point with only one (composite) matrix.
- Much faster for large # of points!
- The composite matrix describing the affine transformation always has the bottom row 0, 0, 0, 1 (2D), or 0, 0, 0, 1 (3D).

Recap: Homogeneous Matrices

Note:
- Multiplication of the matrix with a constant does not change the transformation!
\[
\begin{pmatrix}
  m_{11} & m_{12} & m_{13} & t_x & 0 \\
  m_{21} & m_{22} & m_{23} & t_y & 0 \\
  m_{31} & m_{32} & m_{33} & t_z & 0 \\
  0 & 0 & 0 & 1 & 1
\end{pmatrix}
\begin{pmatrix}
  x' \\
  y' \\
  z' \\
  1
\end{pmatrix}
= T
\begin{pmatrix}
  x \\
  y \\
  z \\
  1
\end{pmatrix}
\]

Recap: Homogeneous Vectors

Representing vectors in homogeneous coordinates
- Need representation that is only affected by linear transformations, but not by translations.
- This is achieved by setting \( w = 0 \)
\[
\begin{pmatrix}
  x \\
  y \\
  z \\
  0
\end{pmatrix}
= T
\begin{pmatrix}
  x \\
  y \\
  z \\
  1
\end{pmatrix}
\]

Recap: Homogeneous Coordinates

Properties
- Unified representation as 4-vector (in 3D) for
  - Points
  - Vectors / directions
- Affine transformations become 4x4 matrices
  - Composing multiple affine transformations involves simply multiplying the matrices.
- 3D affine transformations have 12 degrees of freedom.
- Need mapping of 4 points to uniquely define transformation.

The Rendering Pipeline

Geometry Processing
- Geometry Database
  - Model/View Transform
  - Lighting
  - Perspective Transform
  - Clipping
- Scan Conversion
  - Texturing
  - Depth Test
  - Blending
  - Framebuffer

Modeling Transformation

Purpose:
- Map geometry from local object coordinate system into a global world coordinate system.
- Same as placing objects.

Transformations:
- Arbitrary affine transformations are possible
  - Even more complex transformations may be desirable, but are not available in hardware.
  - Freeform deformations.
**Viewing Transformation**

**Purpose:**
- Map geometry from world coordinate system into camera coordinate system
- Camera coordinate system is right-handed, viewing direction is negative z-axis
- Same as placing camera

**Transformations:**
- Usually only rigid body transformations
  - Rotations and translations
- Objects have same size and shape in camera and world coordinates

---

**Model/View Transformation**

**Combine modeling and viewing transform.**
- Combine both into a single matrix
- Saves computation time if many points are to be transformed
- Possible because the viewing transformation directly follows the modeling transformation without intermediate operations

---

**Rendering Geometry in OpenGL**

```c
glBegin(GL_TRIANGLES);
  glVertex3f(x1, y1, z1); // vertex 1 of triangle 1
  glVertex3f(x2, y2, z2); // vertex 2 of triangle 1
  glVertex3f(x3, y3, z3); // vertex 3 of triangle 1
  glVertex3f(x4, y4, z4); // vertex 1 of triangle 2
  glVertex3f(x5, y5, z5); // vertex 2 of triangle 2
  glVertex3f(x6, y6, z6); // vertex 3 of triangle 2

glEnd();
```

---

**Rendering Geometry in OpenGL**

**Additional attributes**
- glColor3f: RGB color value (0..1 per component)
- glNormal3f: normal vector
- glTexCoord2f: texture coordinate (explained later)

**OpenGL is state machine:**
- Every vertex gets color, normal etc. that corresponds to last specified value

---

**Rendering Geometry in OpenGL**

**Example:**

```c
glBegin(GL_TRIANGLES);
  glColor3f(1.0, 0.0, 0.0);
  glVertex3f(1.0, 0.0, 0.0);
  glColor3f(0.0, 0.0, 1.0);
  glVertex3f(0.0, 1.0, 0.0);
  glColor3f(0.0, 0.0, 0.0);
  glVertex3f(0.0, 0.0, 0.0);

glEnd();
```

---

**OpenGL Naming Scheme**

**Function names:**

```
       gl Vertex 3 f
         \\
        \\
Operation  Dimensionality  Type of parameters
OpenGL Prefix  4  e.g. f (float)  d (double) i (integer)
Missing coordinates are 0 (x, y, z) or 1 (w)
```

---

4
Matrix Operations in OpenGL

**2 Matrices:**
- ModelView matrix $M$
- Projective matrix $P$

**Example:**

```c
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
```

```c
glTranslatef( 4, 3 );
glRotatef( 30, 0.0, 0.0, 1.0 );
glTranslatef( -4, -3 );
glBegin( GL_TRIANGLES );
```

---

Specification matrices (replacement)
- `glLoadIdentity()`
- `glLoadMatrixf( GLfloat *m )` // 16 floats

**Specification matrices (multiplication)**
- `glMatrixMode( GL_MODELVIEW )` // 16 floats
- `glTranslatef( GLfloat x, GLfloat y, GLfloat z )` // angle and axis
- `glScalef( GLfloat x, GLfloat y, GLfloat z )`
- `glTranslatef( GLfloat x, GLfloat y, GLfloat z )`

---

Interpreting Composite OpenGL Transformations

**Example for earlier lectures:**
- Rotation around arbitrary center
- In OpenGL:

```c
// initialization of matrix
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
glTranslatef( 4, 3 );
glRotatef( 30, 0.0, 0.0, 1.0 );
glTranslatef( -4, -3 );
glBegin( GL_TRIANGLES );
```

---

Matrix Operations in OpenGL

**Semantics:**
- `glMatrixMode` sets the matrix that is to be affected by all following transformations (multiplication from the right)
- Transformations that affect a vertex first have to be specified last
- Whenever primitives are rendered with `glBegin()`, the vertices are transformed with whatever the current modelview and perspective matrix is
  - Normals are transformed with the inverse transpose

---

Matrix Operations in OpenGL

**Perspective Matrices (details next lecture):**
- `glFrustum( left, right, bottom, top, near, far )`
  - Specifies perspective xform (near, far are always positive)
- `glOrtho( left, right, bottom, top, near, far )`

**Convenience Functions:**
- `gluPerspective( fovy, aspect, near, far )`
  - Another way to do perspective
- `gluLookAt( eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ )`
  - Useful for viewing transform

---

Transformation Hierarchies

**Scene may have a hierarchy of coordinate systems**
- Stores matrix at each level with incremental transform from parent's coordinate system

**Scene graph**

- `load` in root
- `stripe1`, `stripe2`...
- `car1`, `car2`...
- `w1`, `w2`, `w3`, `w4`...

---


**Transformation Hierarchy Example 1**

- World
  - Torso
    - LLeg
    - RLeg
    - LArm
    - RArm
    - Head
  - Left
  - Right
  - LHand
  - RHand

Transformation:
trans((0.3, 0.0, 0.0)) rotz(0)

---

**Transformation Hierarchies**

- Hierarchies don’t fall apart when changed
- Transforms apply to graph nodes beneath

---

**Brown Applets**


- Have a look later

---

**Transformation Hierarchy Example 2**

- Draw same 3D data with different transformations: instancing

---

**Matrix Stacks**

**Challenge of avoiding unnecessary computation**

- Using inverse to return to origin
- Computing incremental $T_1 \rightarrow T_2$

Object coordinates

World coordinates

---

**Matrix Stacks**

```
D = C * scale(2, 2, 2) * trans(1, 0, 0)
```

- `glPushMatrix()`
- `glPopMatrix()`
- `glPushMatrixStack()`
- `glPopMatrixStack()`
- `glScalef(2, 2, 2)`
- `glTranslatef(1, 0, 0)`
- `glDrawArrays()`
- `glPopMatrixStack()`
Modularization

**Drawing a scaled square**
- Push/pop ensures no coord system change

```c
void drawBlock(float k) {
    glPushMatrix();
    glScalef(k, k, k);
    glBegin(GL_LINE_LOOP);
    glVertex3f(0, 0, 0);
    glVertex3f(1, 0, 0);
    glVertex3f(1, 1, 0);
    glVertex3f(0, 1, 0);
    glEnd();
    glPopMatrix();
}
```

Matrix Stacks

**Advantages**
- No need to compute inverse matrices all the time
- Modularizes changes to pipeline state
- Avoids incremental changes to coordinate systems
- Accumulation of numerical errors

**Practical issues**
- In graphics hardware, depth of matrix stacks is limited
  - Typically 16 for model/view and about 4 for projective matrix

Transformation Hierarchy

**Example 3**
- `glLoadIdentity();`
- `glTranslatef(1.0, 0.0, 0.0);`
- `glPushMatrix();`
- `glRotatef(90.0, 0.0, 1.0);`
- `glTranslatef(0.0, 2.0, 0.0);`
- `glScalef(2.1, 1.1, 1.0);`
- `glTranslatef(1.0, 0.0, 0.0);`
- `glPopMatrix();`

**Example 4**
- `glTranslatef(x,y,z);`
- `glRotatef(θx, θy, θz);`
- `glPushMatrix();`
- `glPushMatrix();`
- `glTranslatef(1.5, 0.7, 0.0);`
- `glDrawHead();`
- `glPopMatrix();`
- `glPopMatrix();`
- `glPushMatrix();`
- `glRotatef(θ1, θ2, θ3);`
- `glPushMatrix();`
- `... (draw other arm)`

Hierarchical Modeling

**Advantages**
- Define object once, instantiate multiple copies
- Transformation parameters often good control knobs
- Maintain structural constraints if well-designed

**Limitations**
- Expressivity: not always the best controls
- Can't do closed kinematic chains
  - Keep hand on hip

Single Parameter: simple

**Parameters as functions of other params**
- Clock: control all hands with seconds
  - \[ m = s/60, h = m/60, \]
  - \[ \theta_s = (2 \pi s) / 60, \]
  - \[ \theta_m = (2 \pi m) / 60, \]
  - \[ \theta_h = (2 \pi h) / 60 \]
Single Parameter: complex

Mechanisms not easily expressible with affine transforms

http://www.flying-pig.co.uk

Coming Up:

Friday:
- Triangle strips/fans
- Perspective projection

Next Week:
- Perspective projection
- Lighting/shading